Immersive Visualization of Supercomputer Applications: Exploring the End-to-End Lag of Different Applications

Valerie E. Taylor *Jian Chen*
EECS Department
Northwestern University

Terrence L. Disz *Michael E. Papka* *Rick Stevens*
Mathematics and Computer Science Division
Argonne National Laboratory

Abstract

Virtual reality has been demonstrated to be very useful in the visualization of science and the synthesis of vast amounts of data. Further, advances in high-speed networks make feasible the coupling of supercomputer simulations and virtual environments. The critical performance issue to be addressed with this coupling is the end-to-end lag time (i.e., the delay between a user input and the result of the input). In this article we quantify the lag times for four different applications. These applications range from coupling virtual environments with supercomputers to coupling multiple virtual environments for collaborative design. The lag times for each application are given in terms of rendering, network, simulation, tracking, and synchronization times. The results of the survey indicate that for some applications it is feasible, in terms of lag times, to couple supercomputers with virtual environments and to couple multiple virtual environments. For the applications with intolerable lag times, the simulation time dominates the lag time.

1 Introduction

The use of immersive visualization to view the results of scientific simulation allows observers to understand the data in a natural way, as three-dimensional objects. The three-dimensional virtual environment enables the display of nonvisual physical information, such as temperature, velocity, or electric and magnetic fields. In engineering, this technology may be incorporated into the product design cycle, allowing virtual prototyping and testing of products prior to their physical construction. Hence, interactive, immersive three-dimensional visualization is an important medium for scientific applications.

A critical issue facing interactive virtual environments is the end-to-end lag time, that is, the delay between a user action and the display of the result of that action. The critical component of the lag time is highly application dependent. In this article we quantify the various components of the lag time for various visual supercomputer applications.

Interactive immersive visualization of scientific simulations involves four major subsystems: graphics, user interface, simulation, and communications. The graphics subsystem performs the calculations needed to render the objects used in the display. The user interface consists of the projection screens, projectors, interactive devices, and tracking sensors needed to physically realize the virtual environment. A user interacts with the virtual objects via a head tracker or hand-held wand (similar to a mouse). The simulation subsystem performs the calculations for the scientific phenomenon, which varies from application to application. The resultant data is communicated to the graphics subsystem and then displayed. The simulation calculations may be executed on the same machine that is used for the graphics subsystem or a different machine or multiprocessor. The communications subsystem consists of the networks used to communicate information between the user and the graphics system or the simulation and graphics subsystems or multiple virtual reality (VR) devices.

Like any closed-loop adaptive system, if the time between an interaction and the result of that interaction (or lag) is too great, users find it difficult to maintain fine control over system behavior and complain that the system is unresponsive. Lag has been studied in the context of teleoperated machines, head-mounted displays, and telepresence systems [12, 20]. In experiments on a telemanipulation system, Liu and his colleagues [12] found the allowable end-to-end lag time to be 100 ms (0.1 s) and 1000 ms (1 s) for inexperienced and experienced users, respectively. In [18], the work on lag models was extended to include scientific simulations with interactive immersive environments; the work included an extensive study of a finite element simulation with the simulation subsystem located in Illinois and the graphics and user interface subsystems located in California.

This article analyzes the components of lag for four different applications that use virtual environments. These applications, listed below, range from coupling virtual environments with supercomputers to coupling multiple virtual environments for collaborative design.

- *CALVIN (Collaborative Architectural Layout via Immersive Navigation)*: allows people at different sites to work collaboratively on the design and viewing of architectural spaces. This application was developed at the Electronic Visualization Laboratory (EVL) at the University of Illinois at Chicago by Jason Leigh and Andrew E. Johnson.

- *Automotive Disk Brake*: uses a parallel finite element code to allow users to design and analyze an automotive disk braking system under different conditions. This application was developed by Thomas Canfield at Argonne National Laboratory, Milana Huang at University of Illinois at Chicago, and Valerie Taylor at Northwestern University.

- *BoilerMaker*: allows users to design and analyze the placement of pollution control system injectors in boilers and incinerators. This application was developed by Darin Diachen et al. at Argonne National Laboratory.

- *Monte*: a simple application to calculate π using a parallel Monte Carlo algorithm. This application provides insights into the display rates for a simple scene and the coupling between a supercomputer and a virtual environment. Monte is based on sample code from [8]. It was developed by Terrence Disz et al. at Argonne National Laboratory.

The results of the survey indicate that it is feasible, in terms of the end-to-end lag time, to couple simple scientific applications with virtual environments and couple multiple virtual environments. The survey also provides insight into areas of needed research to make immersive visualization of complex scientific simulations feasible, that is, within the tolerable lag range.

2 Virtual Environment

The lag models developed and analyzed for the four applications are based upon the virtual environment available at Argonne National Laboratory. The user interface and graphics subsystems, as well as the graphics software, are described below. The simulation and communication subsystems are tailored to the given application; these subsystems will be described in §4 for each application.

2.1 User Interface Subsystem

The user interface subsystem consists of the CAVE^{TM} (CAVE Automatic Virtual Environment); this medium creates a wide field of view by rear-projecting stereo images onto three 10×9-foot translucent screens (Figure 1). This display system is one of many rear-projected immersive display systems developed at EVL. Other display systems include the ImmersaDesk^{TM}, the Infinity Wall (co-developed with NCSA and the University of Minnesota), and the CAVEsimulator. The CAVE provides the illusion of data immersion by using visual cues, including wide field of view, stereo display, and viewer-centered perspective. A Silicon Graphics (SGI) Power Onyx computes the stereo display, with a resolution of 1024×768 for each image. The Onyx alternately draws left and

right eye images at 96 Hz, resulting in a rate of 48 frames per second per eye. These images are sent to an Electrohome video projector for display. Infrared emitters are coupled with the projectors to provide a stereo synchronization signal for a CrystalEyes LCD glasses worn by each user. These glasses have a sampling rate of 96 Hz that is matched to the projection system; the eyes and brain fuse the alternate left and right eye images to provide stereo views.

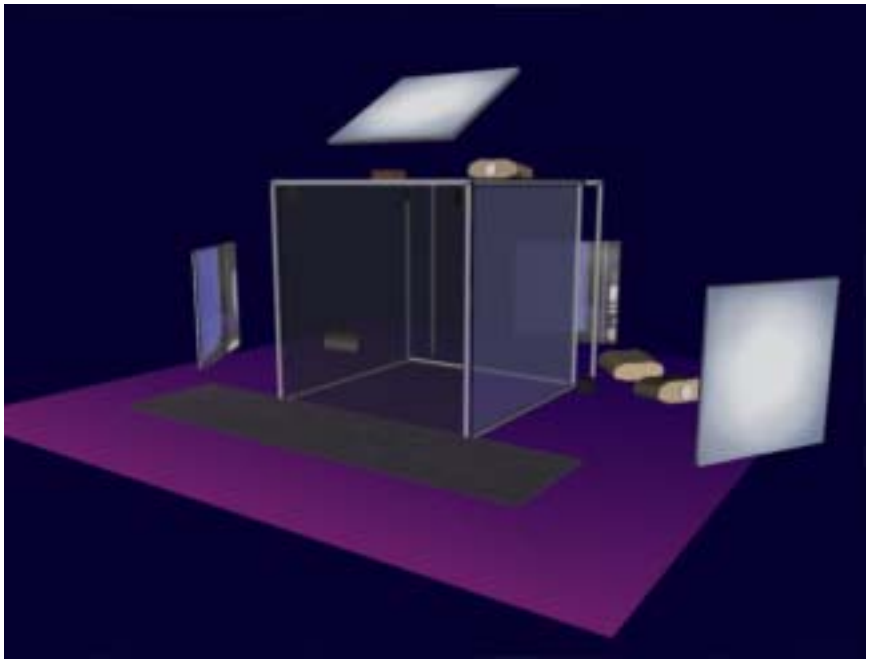

Figure 1: CAVE Virtual Environment (Milana Huang, EVL, 1994)

Tracking is provided by an Ascension Flock of Birds unit with two inputs. One sensor tracks head movements; the other tracks a hand-held wand. These sensors capture position and orientation information on head and wand movements at a rate of 10 to 144 Hz [16]. The existing system is configured to operate at 100 Hz. The buttons on the wand are sampled at a rate of 100 Hz. The location and orientation of the head sensor are used by the SGI Onyx to render images based on the viewer's location in the virtual world. Hence, subtle head movements result in slightly different views of the virtual objects, consistent with what occurs in reality. The wand has three buttons and a joystick that are connected to the SGI Onyx via an IBM PC, which provides A/D conversion, debounce, and calibration. Other observers can passively share the virtual reality experience by wearing LCD glasses that are not tracked.

2.2 Graphics Subsystem

The graphics subsystem consists of the SGI Onyx, which is a shared-memory multiprocessor with a high-performance graphics subsystem. The system used in our experiments has 256 MB of memory, 6 GB of disk, four R4400 250 MHZ processors, and three RealityEngine2 graphics pipelines. Each RealityEngine2 has a geometry engine consisting of Intel i860 microprocessors, a display generator, and 4 MB raster memory [17]. The Onyx is used to drive the virtual environment interface (as discussed above). The CAVE attaches each RealityEngine2 graphics pipeline to a Electrohome Marquee 8000 high-resolution projector to project images onto the left and front screens and the floor.

2.3 Graphics Software

The visualization software, which uses the CAVE Library, consists of one main process that spawns four other processes—three rendering processes and one tracker process—as illustrated in Figure 2. The main process communicates user orientation and inputs to the simulation and makes the data from the simulation available to the rendering processes. The three rendering processes perform the calculations for the user-specified graphics and create an image from the graphics models for the three walls of the CAVE. These three processes are essentially identical with the exception of Render0 process, which reads the tracking data from the tracking process and makes it available to the other rendering processes. Only one rendering process performs this task, to ensure that all three rendering processes perform calculations in response to the same tracker data. These three processes synchronize at the end of each frame update. The tracker process is responsible for continuously reading the tracking information from the serial SGI ports, scaling the data, and writing the data into a region of memory for reference by Render0. All five processes operate asynchronously.

3 End-to-End Lag

The lag time of a virtual environment is highly dependent on the application. There are, however, standard components of the end-to-end lag for a virtual environment coupled with scientific simulations [19, 20]. These components consist of the following: tracking, rendering, simulation, synchronization, frame rate induced, and network. They are described in detail below.

The **tracking** lag is the time required to obtain the position and orientation of the user with the tracked devices. This lag time is dependent on the tracking device used, the number of devices,

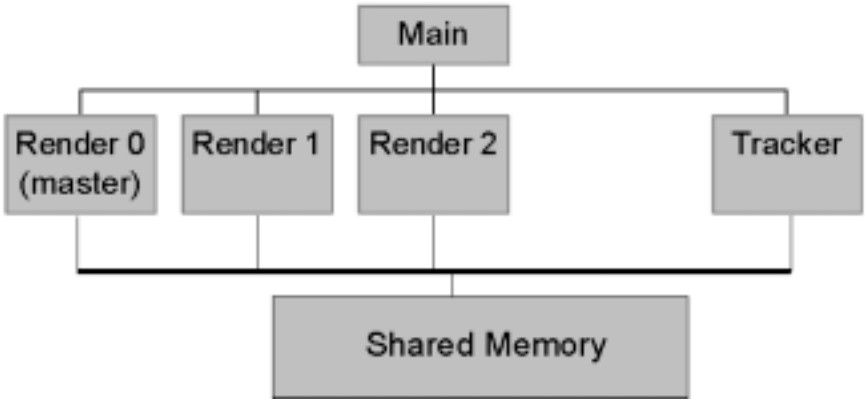

Figure 2: CAVE Software Environment

and the configuration of the devices. Examples of some sensor-based devices include the Ascension Flock of Birds sensors, the Polhemus Fastrak sensors, and the Ascension Bird sensor. The lag for the different devices can range from approximately 7 ms to 120 ms [13].

The **rendering** lag includes both the rendering time (i.e., the time to create an image from a polygonal model) and the time to create the models as dictated by the application. This is consistent with the rendering process of the visualization software described in the preceeding section. This lag time is highly dependent on the scene complexity (e.g., number of objects, number of polygons per object, background) and the user perspective. The variance of the rendering lag is generally large because of the dependence on the user perspective. The maximum tolerable lag is approximately 100 ms (or minimum refresh rate 10 frames/s). If the lag is beyond 100 ms, the viewer begins to perceive a flickering of the images and can no longer fuse the displays into one image [3].

The **simulation** lag is the time required to simulate the scientific phenomenon. This lag time is dependent on the complexity of the simulation and the size of the data set. When a parallel machine is used, the simulation lag is also dependent on the number of processors used for the simulation and the scalability of the parallel algorithms. Highly scalable algorithms result in a decrease in simulation lag with the use of more processors.

The **synchronization** lag is the amount of time data is available between processing stages. This time is dependent upon the occurrence of the sampling points in the various processes and the time required for the various processes. Synchronization lag is inversely proportional to the throughput rates of the various processes. The synchronization time can vary throughout the execution of an application as a result of high variance of the execution time of some of the processes.

The **frame rate induced** lag is the time that the display is out of date. The moment a new frame is displayed, the data on the screens is up to date. As time progresses and the display is not updated, the displayed data becomes more and more out of date. This lag time is dependent only on the frame rate of the projectors.

The last lag component, **network** lag, is the time required to transmit data to and from processes executed on different machines. The network lag may occur between display devices located at different sites or between the simulation process executed on a machine different from the graphics process. This lag time is dependent upon the available network bandwidth and latency, the amount of data to be transferred, and the frequency of the data transfers. For the case of shared networks, a large variance may occur with network lag.

Recall that our virtual environment has two input devices, the head tracker and the wand. Therefore, we consider two classifications of end-to-end lag:

- *Movement of the head tracker.* This type of interaction causes a change to the field of view; the data sent to the simulation process are not modified. This lag is defined as Q_{view}.
- *Movement and clicking of wand buttons.* This type of interaction can change the field of view or the simulation process, dictated by the code-defined wand buttons and joystick. In this paper we focus on wand causing modifications to the simulation process to distinguish the two lags. This lag is defined as $Q_{interact}$.

Lag Equations

The components of the two lags, Q_{view} and $Q_{interact}$, are dependent upon the visualization environment and the application. In this section we provide the general equations for Q_{view} and $Q_{interact}$ based upon the environment at Argonne National Laboratory and the CAVE library as described in §2.

Both lags, Q_{view} and $Q_{interact}$, are affected by the tracker process, that is, the time required to obtain the head or wand positions. Assuming we have a head position, which affects only Q_{view}, this data is placed in shared memory and used by the rendering processes to model objects and render an image (rendering lag or T_{render}). There is likely to be some delay between when the data is available by the tracking process and when it is used by the rendering process; this delay is a synchronization lag, $T_{sync(TR)}$. $T_{sync(TR)}$ is approximated as half the tracking time, since the tracking process runs at a higher frequency than a rendering process. After rendering, the image is placed in the frame buffer and available for display. Some time is required to update the

display, thereby incurring frame-rate induced lag or $T_{sync(F)}$. The resultant equation for Q_{view} is the following:

$$Q_{view} = T_{track} + T_{sync(TR)} + T_{render} + T_{sync(F).} \quad (1)$$

$$(2)$$

The SGI RealityEngine2 graphics pipeline contains a double buffer. The CAVE code places the image data in alternating buffers for each frame. Thus, complete images are displayed in the CAVE.

For $Q_{interact}$, assume we have a wand position, which affects the simulation. Again a delay is likely to occur between when the position data is available from the tracker process and when it is placed in shared memory by Render0. Once in shared memory the data is available to be sent to the simulation process. After sending this data, there is likely to exist a delay between when the data is available in the system buffer for the simulation and when the simulation uses the data. This delay is a synchronization lag, $T_{sync(RS)}$, and approximated as half the simulation time. Simulation occurs, or T_{sim}, followed by the sending of data to the graphics subsystem. There is a likely to be a delay between when the data is available in the system buffer of the graphics subsystem and when it is used by the rendering process, $T_{sync(SR)}$. This lag is approximated as half the rendering time. Then the models are generated and the image created similar to the view lag. The resultant equation for $Q_{interact}$ is the following:

$$\begin{aligned} Q_{interact} = {} & T_{track} + T_{sync(TR)} + T_{sync(RS)} + T_{network} + T_{sim} \\ & + T_{sync(SR)} + T_{render} + T_{sync(F).} \end{aligned} \quad (3)$$

Here $T_{network}$ includes the time to send the data from the graphics subsystem to the simulation and the time to send the data from the simulation to the graphics subsystem.

Equation (2) assumes that the simulation has only one sampling point per iteration. This is consistent with what occurs with the engineering design applications described in the following section, for which accuracy of the display is important. It is possible to allow multiple sampling points in the simulation and progressive updates of the simulation data, similar to progressive refinement that is widely used in the area of graphics [9, 2, 14, 4]. For the case of progressive updates, the display may contain partial information or approximations of the data. The use of progressive updates or multiple sampling points is application dependent.

4 Case Studies

In this section we quantify the end-to-end lag of four CAVE applications and identify the critical components of the lag. The lag times were generated from the performance traces obtained by using the Pablo instrumentation software [15]. The Pablo instrumentation software was shown to incur very little overhead. All the codes (i.e., simulation and CAVE library) were instrumented with Pablo trace cases. Each application was executed multiple times for consistency; each trace is based upon at least 20 minutes of simulation time. All these applications are steered interactively from within the virtual environment of the CAVE.

Recall that the tolerable end-to-end lag time for telemanipulation is in the range of 100 ms to 1000 ms. This is used as the point of reference for all the applications. Further experiments are needed to determine a baseline for application that involve walk-throughs, thereby requiring no telemanipulation.

4.1 Monte Carlo Simulation of π

The calculation of π is a very simple application that demonstrates the the coupling of the supercomputer simulation with the virtual environment. A Monte Carlo method of integration to calculate π was extended from a two-dimensional problem, as outlined in [8], to a three-dimensional problem. The results of the calculations are displayed within the CAVE as points in a sphere (see Figure 3). The white objects in the figure identify the position of the user in the CAVE.

4.1.1 Simulation Component

The simulation component consists of four processes running on 4 nodes of an IBM SP supercomputer. Each SP node has 128 MB of memory and a 1 GB local disk and is connected to other SP nodes via a high-speed network. The Monte Carlo simulation uses a server–worker model to approximate π. One process, the server, generates random numbers that are distributed to the worker processes in chunks. The three worker processes identify whether the random point is inside or outside of the sphere. The value of π is proportional to the fraction of the random points that are within the sphere.

4.1.2 Interconnections

The interconnections consist of the user input display devices and the Ethernet connection between the IBM SP and the SGI Onyx; data is shipped over the Ethernet connection by using the

CAVEcomm library [7]. Control of the simulation is handled via the wand by using one of its buttons. Each button press results in the sending of a message to the IBM SP to request more points for displaying in the CAVE. As the data arrives from the IBM SP, it progressively fills in the volume of a sphere as illustrated in the left picture in Figure 3. The data is sent from all three worker processes.

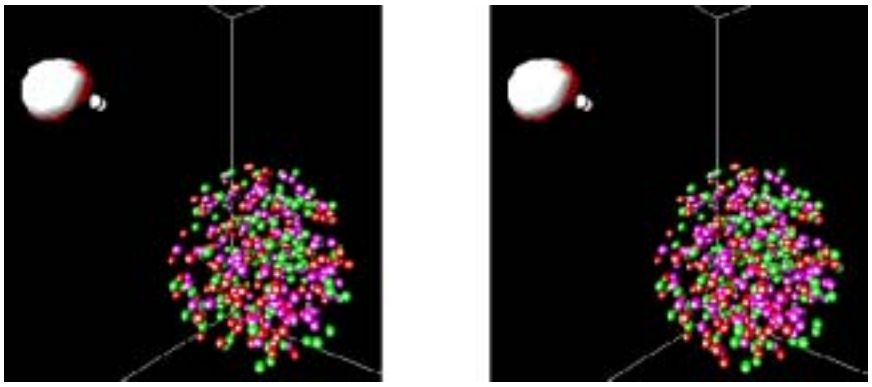

Figure 3: Two Snapshots of the Simulation Filling in Data for the Monte Carlo Simulation of π

4.1.3 Lag Discussion

The mean lag times and percentages are given in Figures 4 and 5, respectively. Equations (2) and (3) are use for the two lags. The results indicate that the two lags Q_{view} and $Q_{interact}$ are within the tolerable 100 ms to 1000 ms range. These results occur because the scene is very simple, as shown in Figure 3, and the simulation is also very simple. Further experiments indicated that a linear relationship exists between number of polygons in a scene and T_{render}. Again, this occurs because of the simplicity of the scene, which has no background; the scene contains only the random points. The amount of data shipped per button press from the simulation to the CAVE is approximately 600 bytes. Hence, for very simple applications it is feasible to couple the supercomputer with the virtual environment via Ethernet and allow for true, real-time interactive analysis.

4.2 Automotive Disk Brake Analysis

This application couples a complex scientific simulation of an automotive disk brake system with the virtual environment; this application is used for engineering analysis. Some critical issues related to automotive disk brakes are the heating and stressing of the material used for the disk and pads, the wear on this material, and the pitch of the sound that occurs when the pads are applied to a rotating disk. The virtual environment interface to the analysis consists of a disk brake in an automobile (see Figure 6).

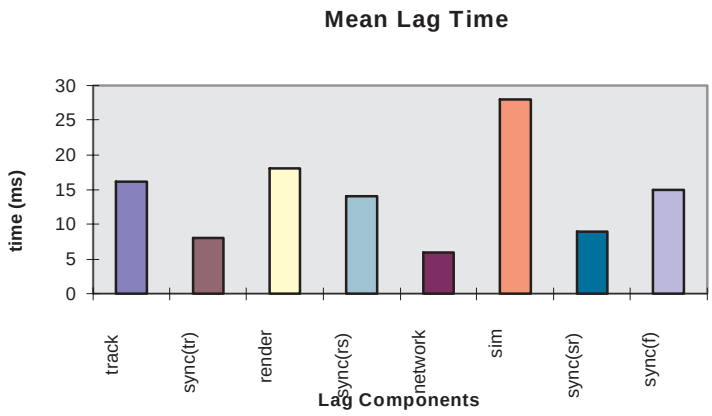

Figure 4: Mean Lag Time

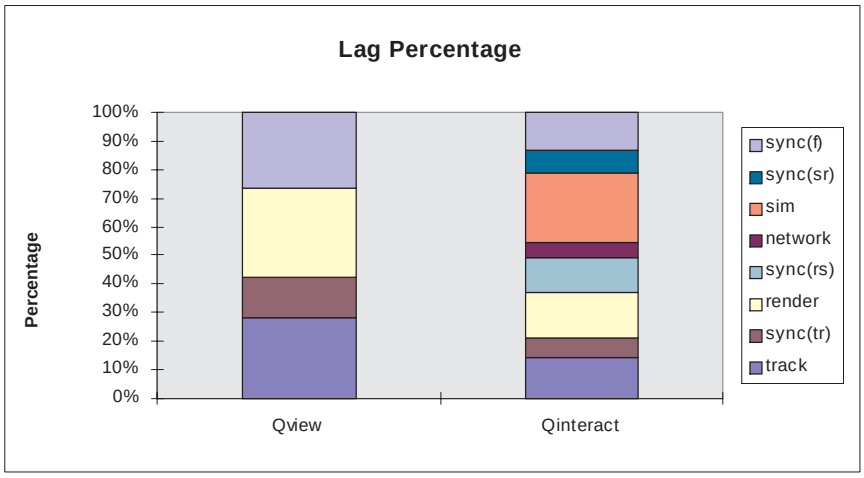

Figure 5: Lag Percentages

Braking times generally are on the order of seconds or tens of seconds. Each time step, which is generally on the order of a few milliseconds, can require tens of minutes of execution time on a single processor. Hence, parallel systems are necessary for this application. Initial transients, such as closing the brake pads, take place in a few milliseconds, whereas the actual braking occurs over the longer time when the pads are clamped on the rotor. Stable-implicit time integration is required to model this long-term braking. The disk brake system is modeled as a finite element problem using the FIFEA (Fast Implicit Finite Element Analysis) code developed at Argonne National Laboratory. The complete disk brake system consists of 3790 elements, 5636 nodes, and 22,544 degrees of freedom (4 degrees of freedom per node).

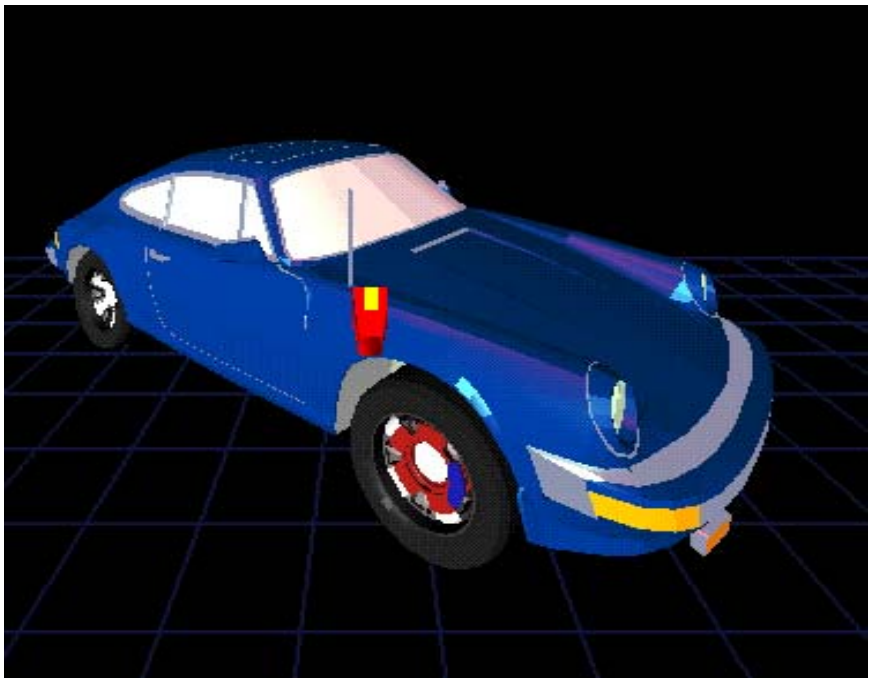

Figure 6: Simulation Showing the Disk Brake in the Front Tire of an Automobile

4.2.1 Simulation Component

The finite element simulation was executed on 8 and 16 processors of the IBM SP. FIFEA makes extensive use of the PETSc (Portable, Extensible, Toolkit for Scientific computation) library [1] to do linear algebra and to manipulate sparse matrices and vectors. FIFEA and PETSc use the MPI library [8] for communication between the IBM SP processors.

4.2.2 Interconnections

Only one IBM SP processor sends and receives results to and from the SGI Onyx over Ethernet via the CAVEcomm library. This one processor executes the FIFEA code on its assigned subdomain in addition to collecting the generated data from all the other processors. Using the head tracker, a scientist can manipulate the virtual environment to focus on various features (for example, areas of high stress or high temperature) of the brake system. Further, while viewing the features, the scientist can use the wand to modify the simulation to test different conditions.

4.2.3 Lag Discussion

Figures 7 and 8 provide the mean lag times and percentage of each lag component, respectively, for the finite element simulation executed on 8 processors. Again Equations (1) and (2) are used for

Q_{view} and $Q_{interact}$. As described above, the brake application uses a significantly more complicated scene than does the Monte Carlo application; this is reflected in the factor of five increase in the rendering time. The results for the total lag times indicate that the view lag is within the tolerable range, but the interaction lag is two orders of magnitude beyond this range. The large simulation time dominates $Q_{interact}$ and results in $T_{sync(RS)}$ also being large. The finite element simulation is significantly more complex than the Monte Carlo simulation; this fact is reflected in the three orders of magnitude difference in simulation time. This finite element simulation is indicative of simulations used in engineering design analyses.

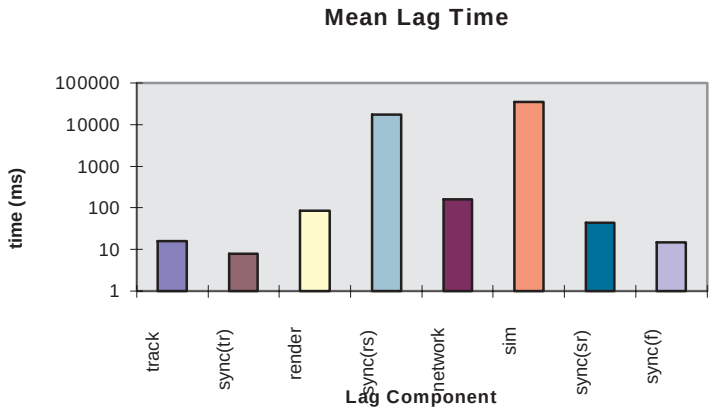

Figure 7: Mean Lag Time (8 processors)

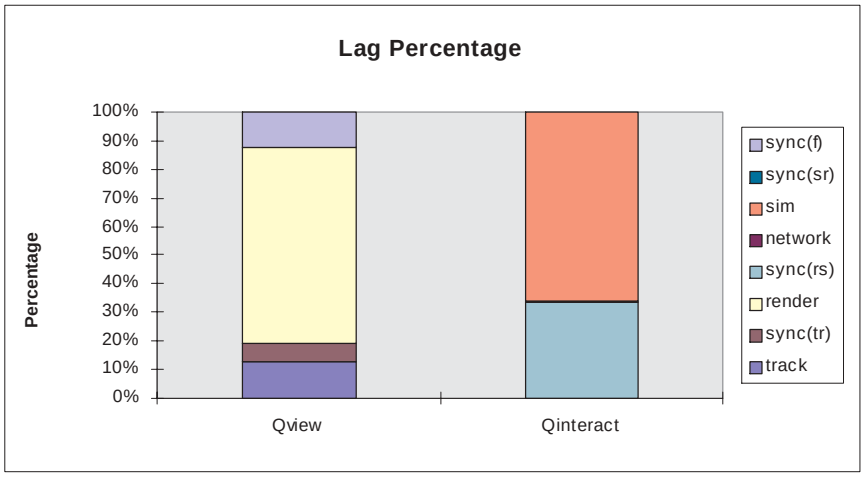

Figure 8: Lag Percentages (8 processors)

The values given in Figures 9 and 9 correspond to the simulation executed on twice as many processors, 16 processors. The use of 16 processors results in a relative speedup (relative to the 8-

processor case) of 1.4. The effect of this speedup is a 29% reduction in $Q_{interact}$. There is still a need to reduce $Q_{interact}$ by approximately two orders of magnitude. To achieve this goal would, however, require the use of at least 800 to 1600 processors. The brake application provides motivation for the need to develop efficient techniques to speed up fixed-size problems.

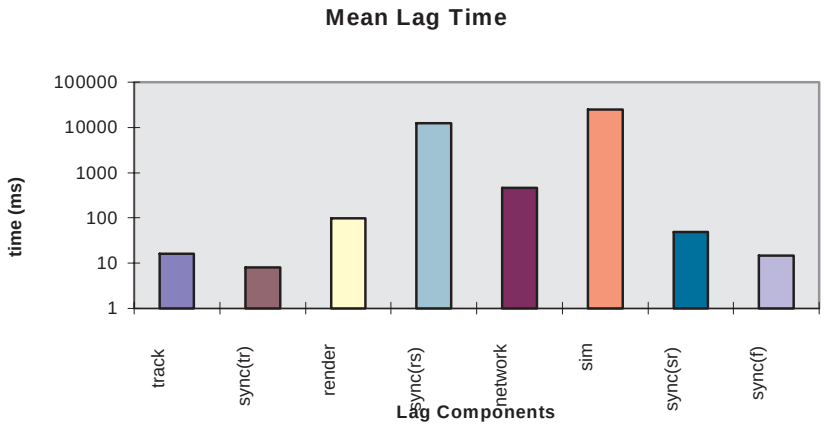

Figure 9: Mean Lag Time (16 processors)

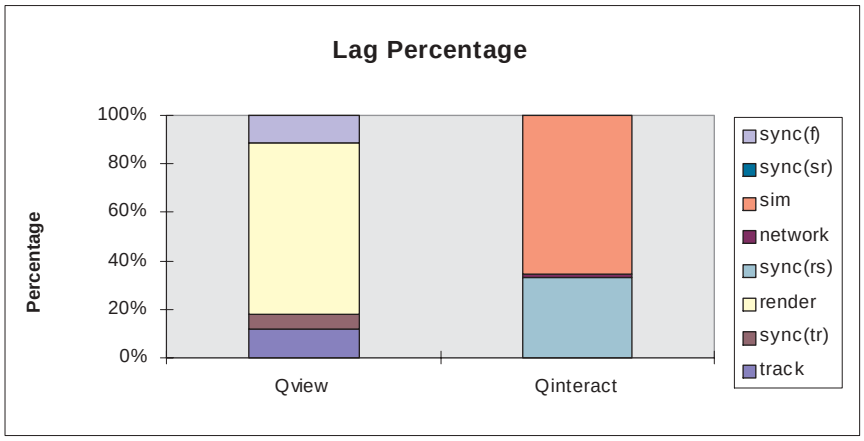

Figure 10: Lag Percentages (16 processors)

It should be noted that there is a small increase in Q_{view} for the 16-processor case as compared with the 8-processor case. This occurred because of the increase in $T_{network}$. Recall the Ethernet is used to interconnect the IBM SP with the SGI Onxy. Ethernet is a shared network, therefore susceptible to bursty traffic.

4.3 BoilerMaker

BoilerMaker is an interactive tool for designing and analyzing the placement of pollution control system injectors in boilers and incinerators [6]. BoilerMaker allows various boiler designs to be imported (a tire incinerator is used for the tests shown in this article) and allows the user to place injector units within the boiler. Once an injector has been placed in the CAVE, the user can see and evaluate the estimated spray coverage of the pollution control chemicals (see Figure 11).

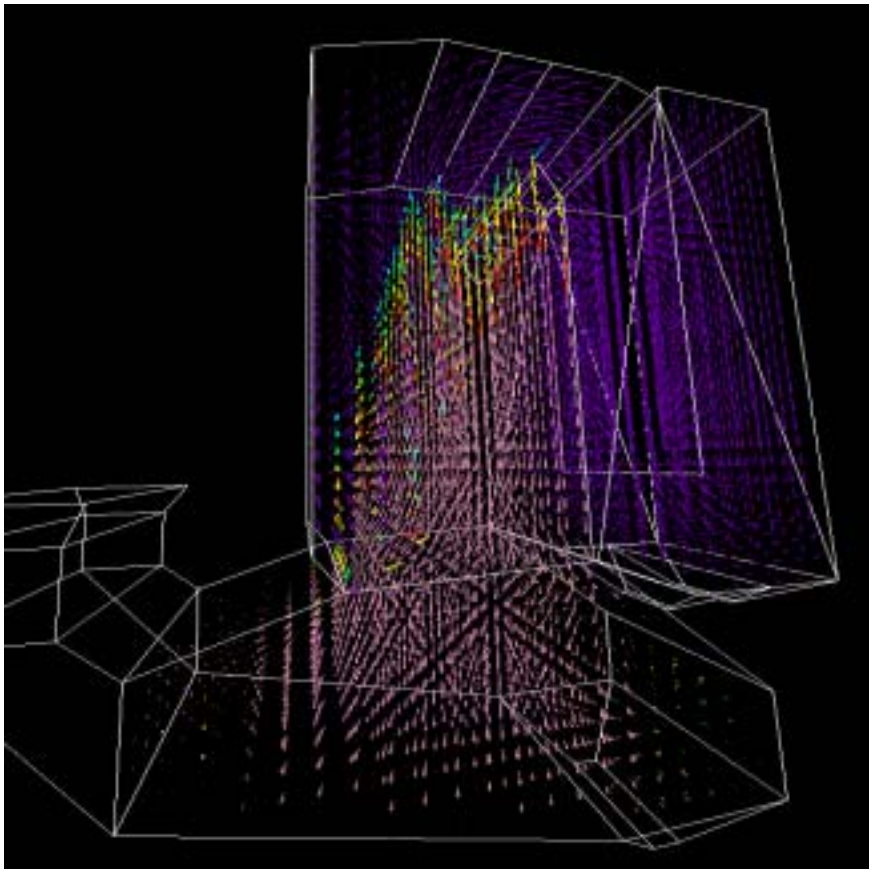

Figure 11: A Wire-Frame Version of the Tire Incinerator Exposing a Vector Field of Gas Flow

4.3.1 Simulation Component

The BoilerMaker application uses precalculated steady-state combustion solutions as a starting state. Users can interactively place streamlines and injectors within the virtual incinerator. An additional SGI is used to generate the data corresponding to the coverage (given by arrows) of the user-placed streamlines of the air-flow within the boiler and injector's spray fields. Further, this SGI also calculates the overall effectiveness of the injectors spray field for pollution reduction.

4.3.2 Interconnections

The CAVE sends and receives information about streamlines and injector placement from a remote process using a modified version of the CAVEcomm library. Using the CAVE's wand, a 3D location can be determined for either the location of a injector or the initialization of a streamline. The simulation SGI sends information to the CAVE in quantities of 20 spray units for the streamlines and 500 spray units for the injectors. The two SGI machines are interconnected via Ethernet.

4.3.3 Lag Discussion

Figures 12 and 13 provide the mean lag times and percentage of each lag component, respectively, for the BoilerMaker application. Again the equations for Q_{view} and $Q_{interact}$ are those given in Equations (1) and (2). The results correspond to users inserting injectors, requiring the generation of 500 spray units. The scene for this application is slightly more complex than the Brakes applications, corresponding to the increase in rendering time. The results of the total lag times indicate that Q_{view} is within the tolerable range, but $Q_{interact}$ is more than an order of magnitude beyond this range because of the large simulation time. Similar to the Brakes application, the large simulation time results in $T_{sync(RS)}$ being large. This time can be reduced using parallel processing.

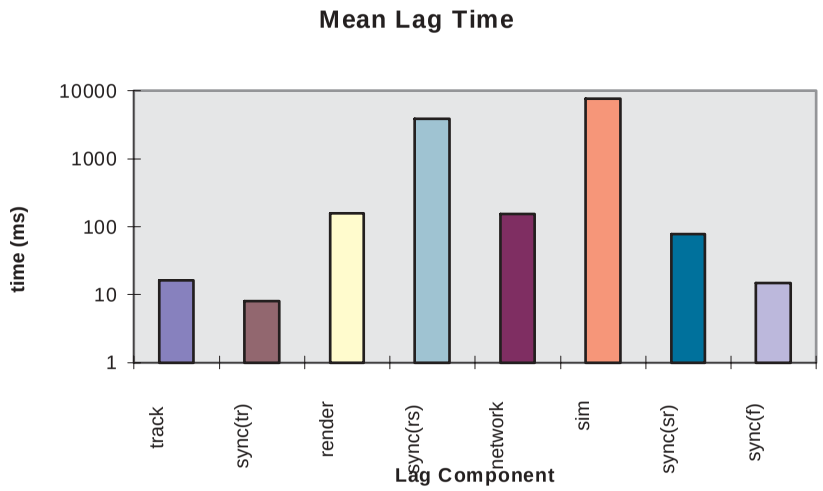

Figure 12: Mean Lag Time (Injector)

Figures 14 and 15 provide the mean lag times and percentage of each lag component, respectively, for user-placed streamlines. Recall that a streamline consists of 20 spray units. The average simulation time is reduced by a factor of approximately 25 as compared with the user-placed in-

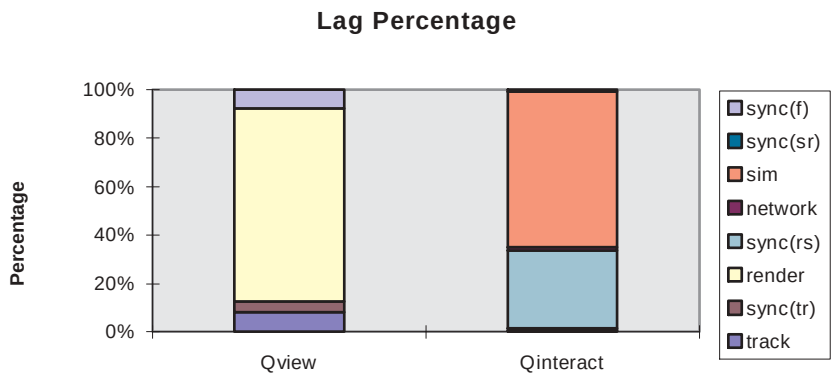

Figure 13: Lag Percentages (Injector)

jector. In addition, the network time is reduced. The result is a substantial reduction in $Q_{interact}$, placing it within the tolerable lag range.

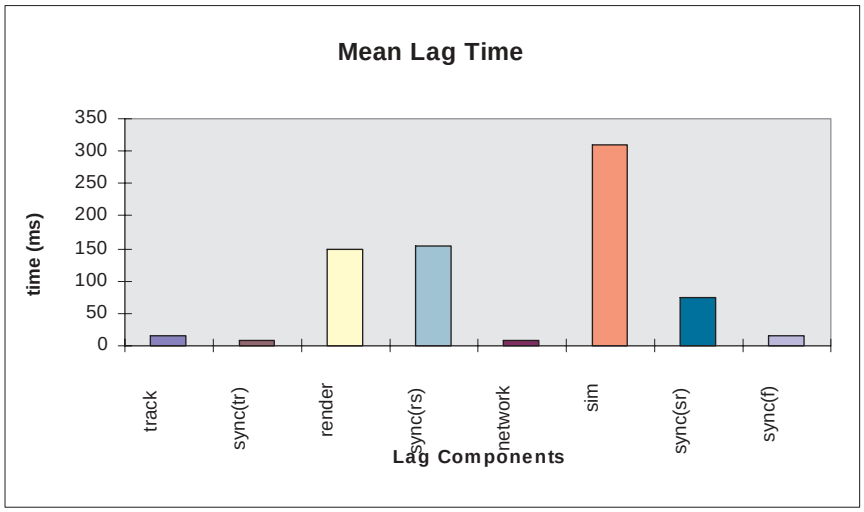

Figure 14: Mean Lag Time (Streamline)

4.4 CALVIN: Collaborative Architectural Layout via Immersive Navigation

CALVIN is a system that provides a shared, collaborative persistent space that allows a number of people to work collaboratively on the design of architectural spaces [10]. Collaborative visualization is achieved by using heterogeneous perspectives, thereby allowing users to assume a variety of different view-points, including inside-out and outside-in (see Figure 16) [11]. CALVIN also incorporates live video and vocal input, both of which are not used in these tests.

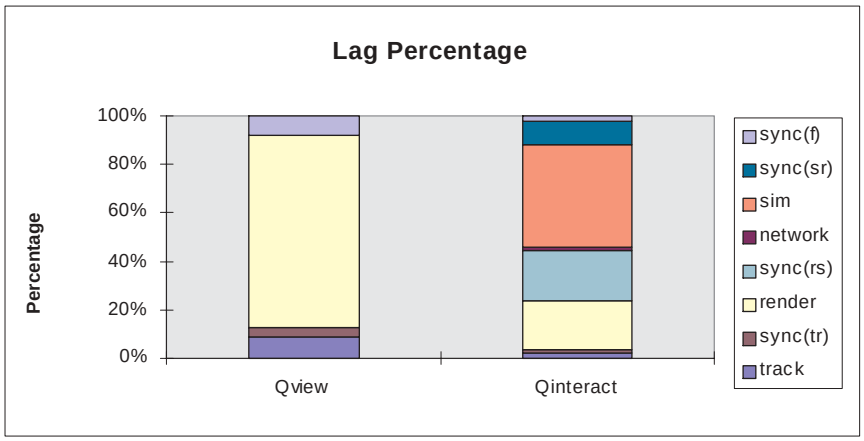

Figure 15: Lag Percentages (Streamline)

4.4.1 Simulation Component

The CALVIN application is designed for simultaneous display on multiple VR devices, located remotely or locally. Each session of CALVIN runs on a separate VR device (where the current VR devices allowed are a CAVE simulator, a CAVE, an ImmersaDesk, or an InfinityWall) interconnected via a central database. No simulations are required; the application entails only modeling and rendering of data based upon a user's perspective. In this paper one CALVIN session used the CAVE and a second session used an ImmersaDesk, interconnected via a local-area network.

4.4.2 Interconnections

Collaboration is achieved by sending and retrieving information to and from a database; this information relates the users' interactions within the scene. The database resides on only one machine, designated as the CALVIN server. The two SGI Onyxes used in the experiments were interconnected via Ethernet. The different machines communicate via a finely tuned sockets library.

4.4.3 Lag Discussion

Figures 17 and 18 provide the mean lag times and percentage of each lag component, respectively, for one CALVIN session in a CAVE. For this application, there is only Q_{view} based upon one user; only one user updates the database. The results indicate that the viewpoint lag is well within the tolerable lag range.

Figures 19 and 20 provide the lag values for two CALVIN sessions, one using the CAVE and and the other one using the ImmersaDesk. The value for $T_{network}$ is an average of the network

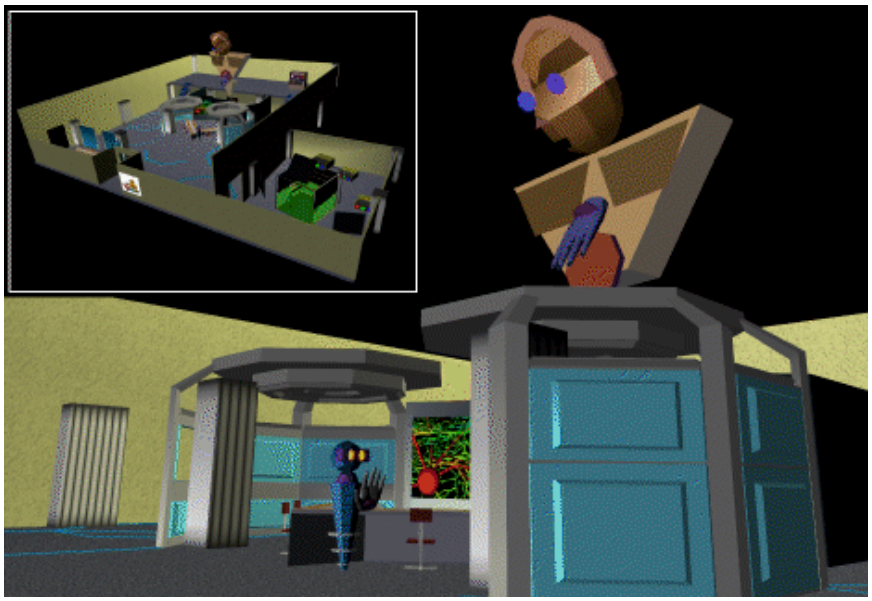

Figure 16: Multiple Viewing Modes of CALVIN (J. Leigh, EVL, 1996)

traffic time. Details about the network traffic for a CALVIN run are given graphically in Figure 21. As expected, there is a large variation in the network traffic, which is highly dependent on the movement of the users. The lag results indicate that even for two display devices, Q_{view} and $Q_{interact}$ are within the tolerable range.

5 Summary

Figure 22 provides a summary of the major lag components for the four different applications. The synchronization lag times are not given because they are proportional to the major lag components — tracker, rendering, network, and simulation. The graph indicates that the largest component of the end-to-end lag for all the applications, with the exception of CALVIN, is the simulation time; the CALVIN application did not require any simulation. This component is especially large for the Boiler with injectors and the two Brakes applications, for which the interaction lag was one to two orders of magnitude beyond the tolerable range. For these three cases, the simulation time are two to three orders of magnitude beyond the other lag times. Hence, for immersive visualization of supercomputer applications to be feasible, significant work must be done to reduce the simulation time to less than one second.

Methods for reducing this simulation time include the use of massively parallel machines (MPPs) and the use of techniques such as progressive updates. To achieve good performance on MPPs, further work is needed to develop highly scalable algorithms for fixed-size problems. Further, for

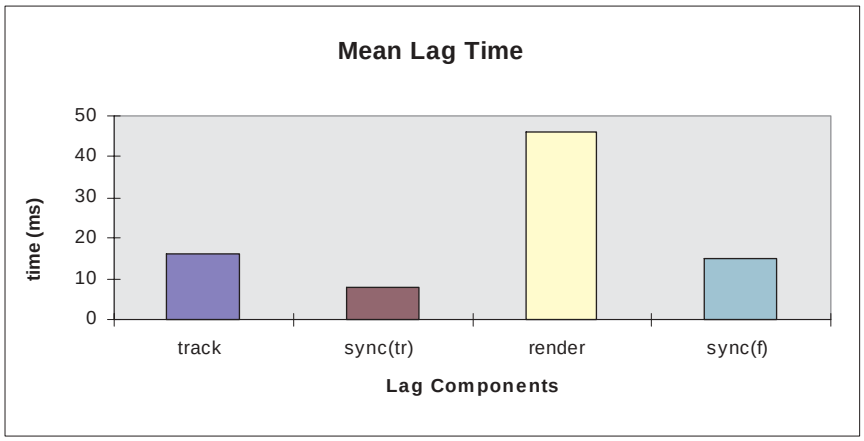

Figure 17: Mean Lag Time (one session)

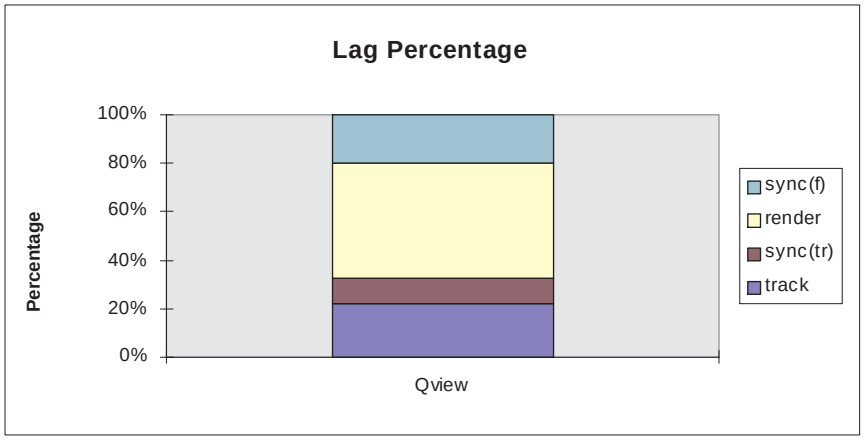

Figure 18: Lag Percentages (one session)

largescale problems, the simulation may require the use of distributed supercomputers to obtain the needed compute power. Hence, the algorithms must be scalable and be able to hide latencies introduced by the resources being distributed.

The results of this survey indicate that it is feasible to couple simple scientific applications, such as the Monte Carlo simulation, with a virtual environment and have the lag be in the range of 0.1 s to 1.0 s. The CALVIN application demonstrates that it is feasible to couple multiple visualization environments to permit collaborative working environments. The survey also provided insight into areas of needed research to make immersive visualization of complex scientific simulations feasible, that is, within the tolerable lag range.

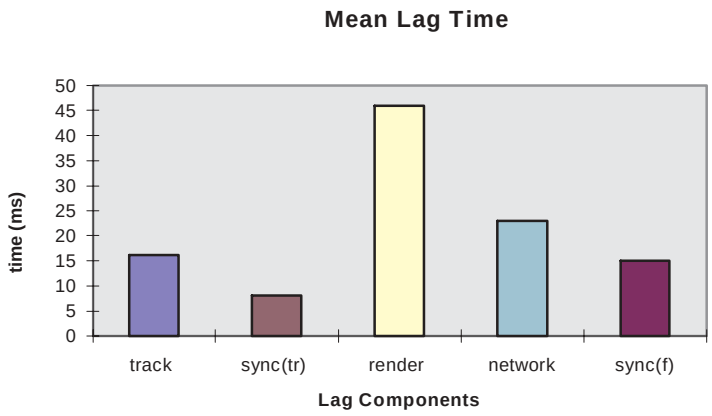

Figure 19: Mean Lag Time (two sessions)

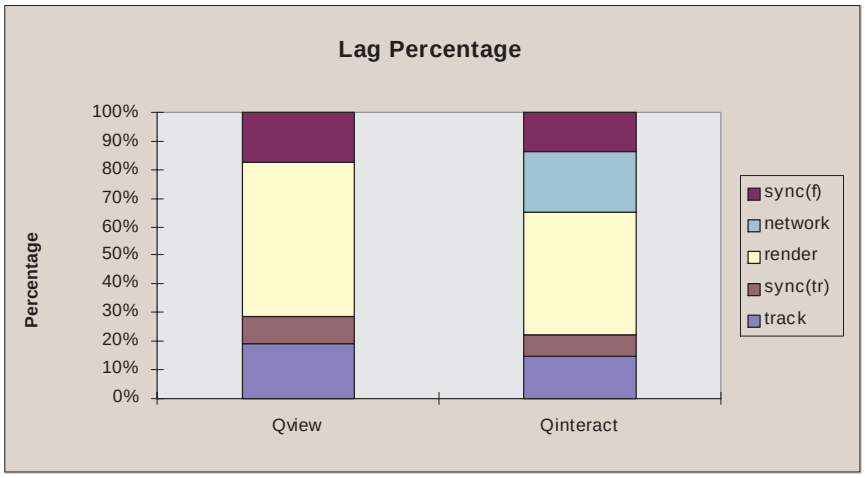

Figure 20: Lag Percentages (two sessions)

Acknowledgments

The authors are grateful to the four groups for donating their codes: the BoilerMaker group Darin Diachin, Lori Freitag, Daniel Heath, Jim Herzog, Bill Michels, and Bob Ryan; the CALVIN group Jason Leigh and Andy Johnson; the Brake group Thomas Canfield, Milana Huang, and Valerie Taylor; and the Monte group Terry Disz, Bill Gropp, Rusty Lusk, Bob Olson, Mike Papka, and Matt Szymanski. Also, the authors acknowledge William Douglas Wilson and Mark Meyer for their assistance with instrumenting the applications.

The authors at Northwestern University were supported by a National Science Foundation Young Investigator Award, under grant CCR–9357781. The authors at Argonne National Laboratory were supported by the Mathematical, Information, and Computational Sciences Division

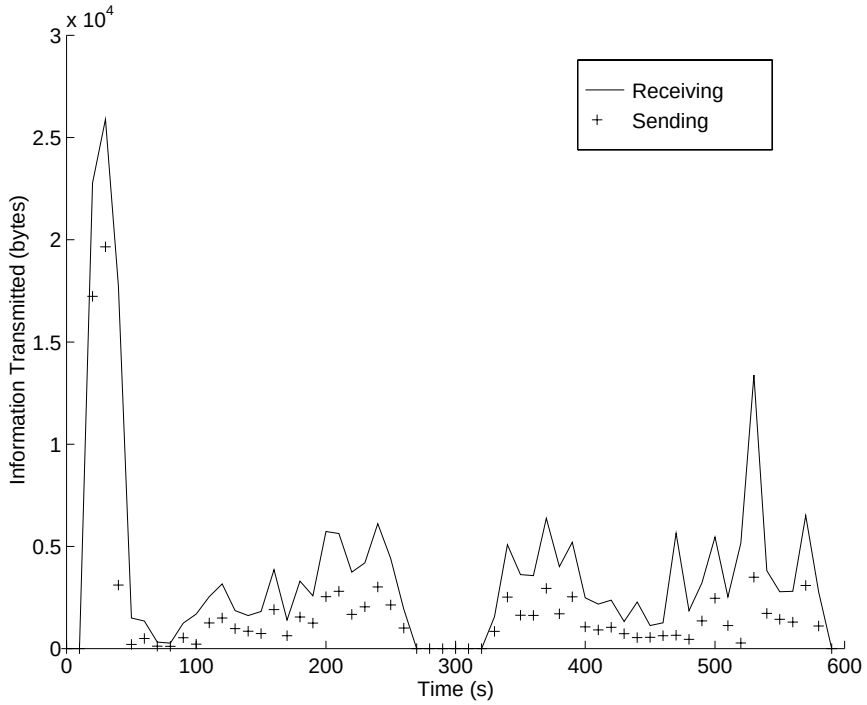

Figure 21: Network Traffic for a Typical CALVIN Session

subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

References

[1] S. Balay, W. Gropp, L. Curfman McInnes, and B. Smith, "PETSc 2.0 Users Manual," Technical Report ANL-95/11, Argonne National Laboratory, 1995.

[2] L. Bergman, H. Fuchs, E. Grant, and S. Spach, "Image Rendering by Adaptive Refinement," *Proceedings of SIGGRAPH*, 1986, pp. 29–37.

[3] G. Burdea and P. Coiffet, *Virtual Reality Techonology*, John Wiley and Sons, Inc., New York, 1994.

[4] M. F. Cohen, S. E. Chen, J. R. Wallace, and D. P. Greenberg, "A Progressive Refinement Approach to Fast Radiosity Image Generation," *Proceedings of SIGGRAPH*, 1988, pp. 75–84.

[5] C. Cruz-Neira, D. J. Sandin, and T. DeFanti, "Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE," *Proceedings of SIGGRAPH*, ACM Press, New York, 1993, pp. 135–142.

[6] D. Diachin, L. Freitag, D. Heath, J. Herzog, W. Michels, and P. Plassmann, "Remote Engineering Tools for the Design of Pollution Control Systems for Commercial Boilers," to appear in the *International Journal of Supercomputer Applications and High Performance Computing.*

[7] T. Disz, M. E. Papka, M. Pellegrino, and R. Stevens, 1995, "Sharing Visualization Experiences among Remote Virtual Environments," *Proceedings of the International Workshop on High Performance Computing for Computer Graphics and Visualization*, 1995, Swansea, Wales, Springer, pp. 217–237.

[8] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI Portable Parallel Programming with the Message-Passing Interface*, MIT Press, Cambridge, Massachusetts, 1994.

[9] A. R. Forrest, "Antialiasing in Practice," in R. A. Earnshaw, ed., *Fundamental Algorithms for Computer Graphics,* NASA ASI Series F: Computer and Systems Sciences, (17), Springer-Verlag, New York, 1985, pp. 113–134.

[10] J. Leigh and A. E. Johnson, "Supporting Transcontinental Collaborative Work in Persistent Virtual Environments," *IEEE Computer Graphics and Applications*, (16)4, July 1996, pp. 47–51.

[11] J. Leigh, A. E. Johnson, and T. A. DeFanti, "CALVIN: An Immersimedia Design Environment Utilizing Heterogeneous Perspectives," *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, IEEE Computer Society, 1996.

[12] A. Liu, G. Tharp, L. French, S. Lai, and L. Stark, "Some of What One Needs to Know about Using Head-Mounted Displays to Improve Teleoperator Performance," *IEEE Transactions on Robotics and Automation*, (9), 1993, pp. 638–648.

[13] M. R. Mine, "Characterization of End-to-End Delays in Head-Mounted Display Systems," *Technical Report TR93-001*, University of North Carolina at Chapel Hill, 1993.

[14] J. Painter and K. Sloan, "Antialiased Ray Tracing by Adaptive Progressive Refinement," *Proceedings of SIGGRAPH*, 1989, pp. 281–288.

[15] D. A. Reed, K. A. Shields, W. H. Scullin, L. F. Tavera, and C. L. Elford, "Virtual Reality and Parallel Systems Performance Analysis," *IEEE Computer*, (28)11, November 1995, pp. 57–67.

[16] *SpacePad Position and Orientation Measurement System Installation and Operation Guide*, Ascension Technology Corporation, Vermont, 1995.

[17] *Silicon Graphics Onyx Installation Guide*, Document Number 108-7042-010, Silicon Graphics Corporation, California.

[18] V. E. Taylor, M. Huang, T. Canfield, R. Stevens, D. Reed, and S. Lamm, "Performance Modeling of Interactive, Immersive Virtual Environments for Finite Element Simulations," to appear in the *International Journal of Supercomputer Applications and High Performance Computing*.

[19] V. Taylor, R. Stevens, and T. Canfield, "Performance Models of Interactive, Immersive Visualization for Scientific Applications," *Proceedings of the International Workshop on High Performance Computing for Computer Graphics and Visualization*, 1995, Swansea, Wales, Springer, pp. 238–252.

[20] M. Wloka, "Lag in Multiprocessor Virtual Reality, " *Presence*, (4), 1995, pp. 50–63.

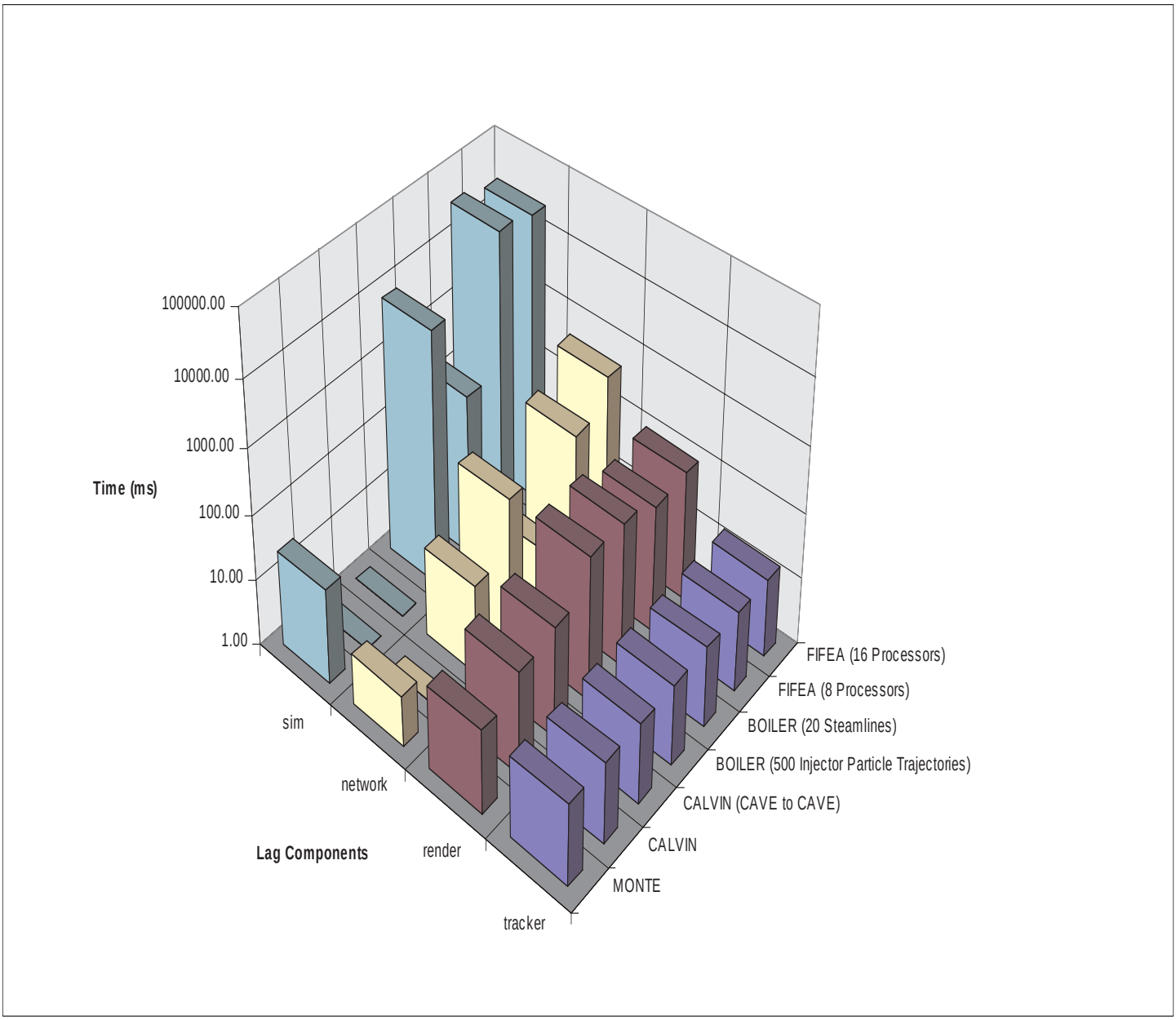

Figure 22: A Comparison of the Four Applications